Reduce Scanning Process for Efficient Mining Tree in Association Rule Mining

Richa Sharma[1], Mr. Premnarayan Arya[2]
S.A.T.I., Vidisha, India[1, 2]

***Abstract*— The essential aspect of mining association rules is to mine the frequent patterns. Due to native difficulty it is impossible to mine complete frequent patterns from a dense database. FP-growth algorithm has been implemented using a Array-based structure, known as a FP-tree, for storing compressed frequency information. Numerous experimental results have demonstrated that the algorithm performs extremely well. But In FP-growth algorithm, two traversals of FP-tree are needed for constructing the new conditional FP-tree. In this paper we present a novel ABFP tree technique that greatly reduces the need to traverse FP-trees and array based FP tree, thus obtaining significantly improved performance for FP-tree based algorithms. The technique works especially well for sparse datasets. We then present a new algorithm which use the FP-tree data structure in combination with the FP- Experimental results show that the new algorithm outperform other algorithm in not only the speed of algorithms, but also their CPU consumption and their scalability.**

Keywords- *：FP-Tree, WSFP –Tree, Frequent Patterns, Array Technique.*

I. INTRODUCTION

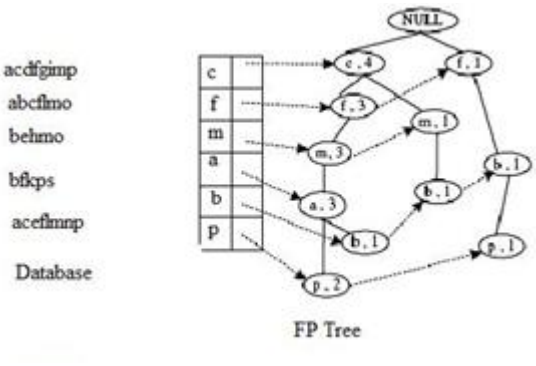
The problem for association rules mining from a data stream has been addressed by many authors but there are several issues (as highlighted in previous sections) that hang about to be addressed. In this part we address the literature review of data stream mining. The work in this domain can be effectively classified into three different domains namely, exact methods for Frequent Item set Mining, Approximate Methods and Memory Management techniques adopted for data stream mining [1,2].

Let I= {i1, i2, in} be a set of items, we call x and I an item set, and we call X a k-item set if the cardinality of item set X is k. Let database T be a multi set of subsets of I, and let support(X) be the percentage of item set Y in T such that X U Y .Informally, the support of an item set procedures how often X occurs in the database. If support(X) + minus , we say that X is a frequent item set , and we denote the set of all frequent item sets by FI. A closed frequent item set is a frequent item set X such that there exists no superset of X with the same support count as X. If X is frequent and no superset of X is frequent, we Say that X is a maximal frequent item set, and we denote the set of all maximal frequent item sets by MFI. [7]

This is the inherent cost of candidate generation, no matter what implementation technique is applied. It is tedious to repeatedly scan the database and check a large set of candidates by pattern matching, which is especially true for mining long patterns. Can one develop a method that may avoid candidate generation-and-test and utilize some novel data structures to reduce the cost in frequent-pattern mining? This is the motivation of this study [6].

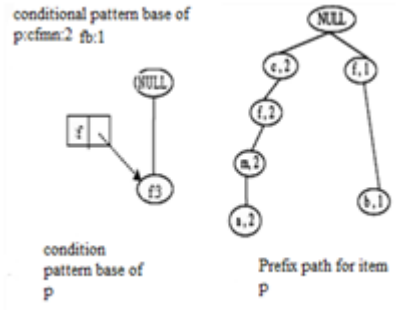
II. RELATED WORK

In the aforementioned FP-growth method [2], a novel data structure, the FP-tree (Frequent Pattern tree) is used. The FP-tree is a compact data structure for storing all necessary information about frequent item sets in a database. Every branch of the FP-tree represents a frequent item set, and the nodes along the branch are ordered decreasingly by the frequency of the corresponding item, with leaves representing the least frequent items. Each node in the FP-tree has three fields: item-name, count and node-link, when item-name registers which item this node represents, count registers the number of transactions represented by the portion for the path reaching this node, and node-link links to the next node in the FP-tree carrying the same item-name, or null if there is none. The FP-tree has a header table associated with it. Single items are stored in the header table in decreasing order of frequency. Each entry in the header table consists of two fields, item-name and head of node-link (a pointer pointing to the first node in the FP-tree carrying the item-name). Compared with Apriority [1] and its variants which need several database scans, the FP-growth method only needs two database scans when mining all frequent item sets. In the first scan, all frequent items are found. The second scan constructs the first FPtree which contains all frequency information of the original dataset. Mining the database then becomes mining the FP-tree. Figure 1(a) shows a database example. After the first scan, all frequent items are inserted in the header table of an initial FP-tree. Figure 1(b) shows the first FP-tree constructed from the second scan. The FP-growth method relies on the following principle: if X and Y are two item sets, the support of item set X UY in the database is exactly that of Y in the restriction of the database to those transactions containing X. This restriction of the database is called the conditional pattern base of X. Given an item in the header table, the growth method constructs a new FP-tree corresponding to the frequency information in the sub-dataset of only those transactions that contain the given item. Figure 2(a) shows the conditional pattern base and the FP-tree for item {p}, this step is applied recursively, and it stops when the resulting smaller FP tree contains only one single path. The complete set of frequent item sets is generated from all single path FP-trees [5]. When adding an item i to the existing item set head, we denote the item set head i by Z, the path from the parent node of this node (node's item-name is i) to the root node in the head's FP-tree is called Z's prefix path. Figure 2(b) shows the prefix paths for item {p}.



(a) (b)

Figure 1. FP-Tree



(a) (b)

Figure 2. Prefix paths for Item

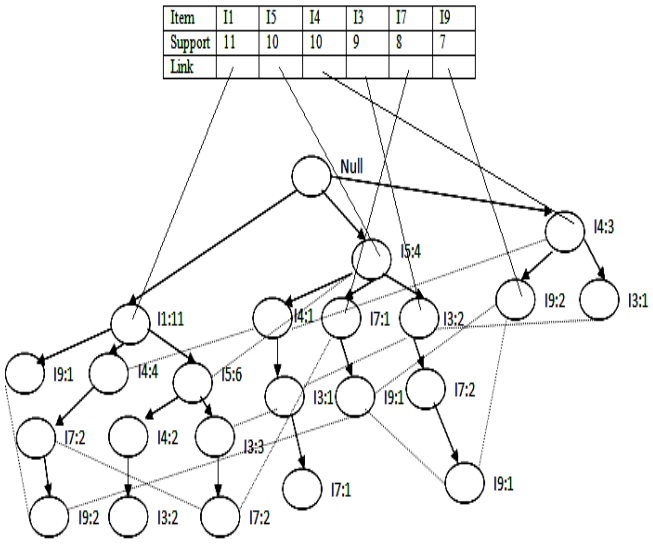


Figure 3. Frequency of Sample Database

III. PROPOSED WORK

Algorithm of WS with Array based technique: Improved FP-tree (IFP-tree) is similar with FP-tree and each node in IFP-tree consists of four fields: item, count, ahead and next. Where item registers which item this node represents, count registers the number of transactions represented by the portion of the path reaching this node, ahead links to the left child or the parent of the node, and next links to the right brother of the node or the next node in IFP-tree carrying the same item, or null if there is none. We also define two arrays: nodecnt and link, and link [item] registers a pointer which points to the first node in the IFP-tree carrying this item, nodecnt [item] registers the support count sum of those nodes in IFP-tree which carry the same item. In comparison with FP-tree, IFP-tree doesn't contain the path from root to leaf-node, contains fewer pointers than FP-tree in mining process, and so may greatly save cost in memory. The construction method of IFP-tree is similar with that of FP-tree, the difference from FP-tree exits in the process of Inserting frequent item sets in each transaction into IFP-tree. In this paper, we don't adopt the method of recursively performing the procedure, insert tree ([p|P], t), but employ a dynamic pointer to complete it.

A. The algorithm constructing IFP-tree as follows:

Procedure FP-tree constructs (T, min_sup)

- *Scan T and count the support of each item, derive a frequent item set (F) and a list (L) of frequent items,in which items are ordered in frequency-descending order*
- *The root of IFP-tree is created and labeled with "root";*
- *For each transaction t UT do*

{

Frequent item set It= t UF, in which items are listed to St according to the order of L, defines a dynamic pointer (p_current) which points to root.

Procedure WSFP-tree constructs (T, min_sup)

- *Scan T and count the support of each item, derive a frequent item set (F) and a list (L) of frequent items, in which items are in sequence of occurrence form;*
- *The root of IFP-tree is created and labeled with "root";*
- *For each transaction t UT do*

{

Frequent item set It= t UF, in which items are listed to St according to the order of occurrence L, defines a dynamic pointer (p_current) which points to root.

- *Traverse IFP-tree in a root-first order and transfer the pointers of ahead and next, count the sum of nodes' support carrying the same item and then list together. For example, let transaction database T be illustrated by Table I, and the minimum support (min_sup) be 4, then we can get the list (L) of frequent items.*

TABLE I. TRANSACTION DATABASE T

A	G	D	C	B
B	D	E	A	M
C	E	F	A	N
A	B	N	O	I
A	C	Q	R	G
A	C	H	I	G
A	F	M	N	O
A	D	B	H	I
J	E	B	A	D
A	K	E	F	C
C	D	L	B	A

Figure 4. Constructed all frequent item set.

TABLE II. TRANSACTION DATABASE T WITH ASCENDING ORDER

A	B	C	D
A	B	D	E
A	C	E	F
A	B	_	_
A	C	_	_
A	C	_	_
A	F	_	_
A	B	D	_
A	B	D	E
A	C	E	F
A	B	C	D

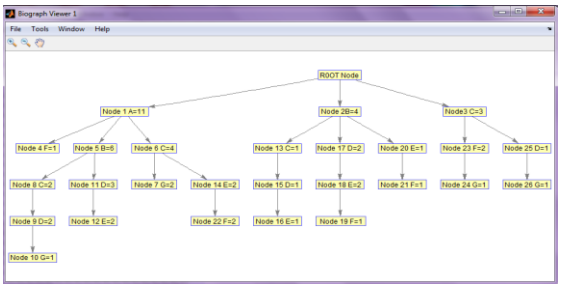


Figure 5. FP Tree constructions

TABLE III. TRANSACTIONAL DATASET

A	G	D	C	B
B	D	E	A	M
C	E	F	A	N
A	B	N	O	I
A	C	Q	R	G
A	C	H	I	G
A	F	M	N	O
A	D	B	H	I
J	E	B	A	D
A	K	E	F	C
C	D	L	B	A

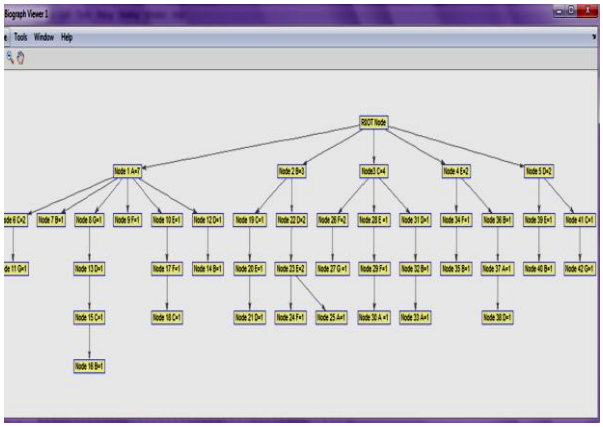


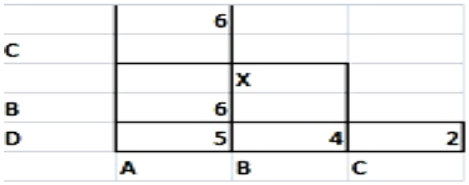
Figure 6. WSFP Tree construction

IV. ARRAY BASED APPROACH:

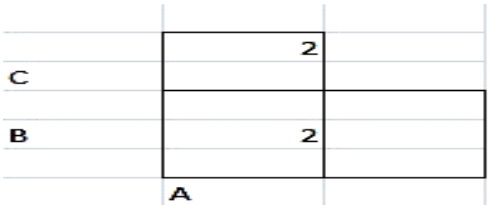
Firstly let an order (<) be the order of the list L, that is, the support descending order of frequent items. Let the letters (i,j,<,k) denote items in database, then i is called the minimum item and k is called the maximum item if i<j<....k. Let the letters (ik,<,i1[<1]) denote items, P be a path from root to the node N in IFP-tree. If there exists a child node N□of the node N and the items (ik,<,i1) appear the sub-path from N to N in order, that is, the item ik corresponds to the node N and i1 corresponds to N,then P is called the path with the array of the item sets{ik,<,i1}, the support count of the node N also is called the base count of A.

The process of building PT (a) is the following: firstly, each node in IFP-tree whose value of item is m is retained in PT (a), the support count of each inner node (except root) is initialized to be zero. Secondly, for each node, we summate the support count of its children.

The main work done in the mining process is traversing the postfix sub-tree to count the support of item sets and constructing new postfix sub-tree, Recall that for each item i of conditional PT(x), two traversals of PT(x) are needed for constructing the new sub-tree PT(k,i). The first traversal finds all frequent items and their counts of support, the second traversal constructs the new sub-tree PT (k, i). In this paper, we use the array technique presented by reference [2]. All cells in the array are initialized as 0.



(A)



(B)

Figure 7. Two Array Examples

During the second scan each transaction, first all frequent items in the transaction are extracted. Suppose these items form item set I, to insert I into PT, the items in I are sorted according to the order in the header of PT. When we insert I into PT, at the same time AU[i,j] is incremented by 1 if {i,j} is contained in I. After the second scan, array A keeps the counts of all pairs of frequent items, as shown in Fig 4.

V. EXPERIMENTAL EVALUATION

The experiments were conducted on 2.4 GHz Pentium with 512 MB of memory running Microsoft Windows XP. All codes were compiled using Matlab7.10. We used Connect-4 downloaded form a website to test and compared FP tree with WSFP tree, which is a real and dense dataset. Fig 8 and Fig 9 shows the experimental results. Here we can see that ABWSFP outperforms WSFP for high levels of minimum support, but it is slow for very low levels.

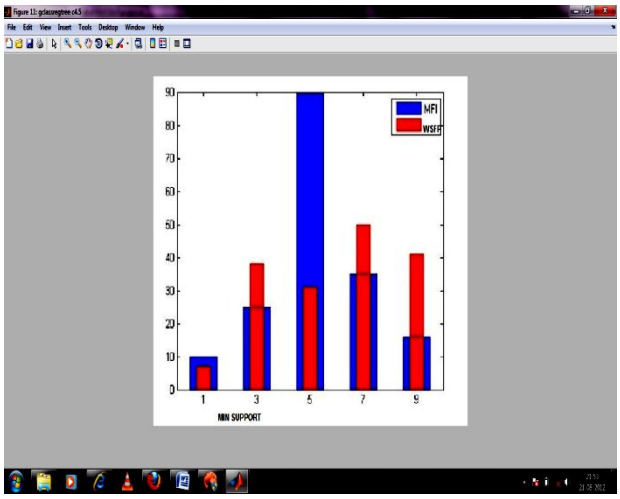


Figure 8. Graphical Representation of Calculated results

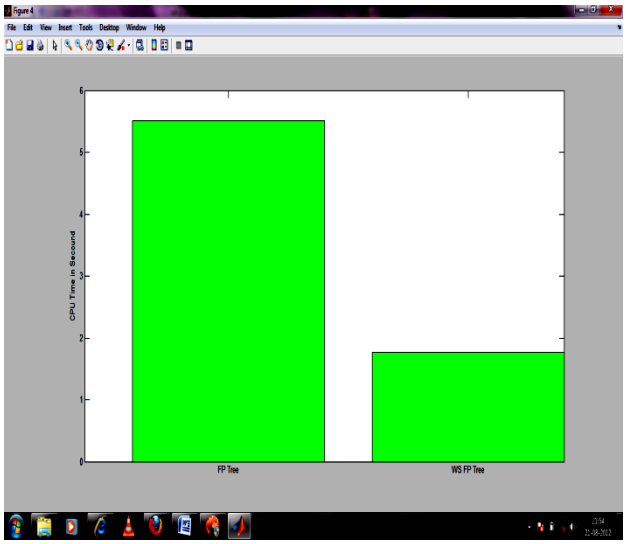


Figure 9. CPU Utilization

VI. CONCLUSIONS

In this paper, an efficient algorithm, called ABWSFP-max, for mining maximal frequent patterns based on improved FP-tree and array technique is proposed, the algorithm improves the conventional FP-tree and by introducing the concept of the array sub-tree, avoids generating the maximal frequent candidate patterns in mining process and therefore greatly reduces the memory consume, it also uses an array-based technique to reduce the traverse time to the improved FP-tree. Therefore it greatly improves the mining efficiency in time and space scalability. Experimental results show that it possesses high mining efficiency and scalability.

REFERENCES

[1] Karun Verma," a better approach to mine frequent item sets using apriori and fp-tree approach"2011.

[2] Implementation of array based technique to improvise representation of fp-tree using iafp-max algorithm", Journal of Global Research in Computer Science (JGRCS) 2011.

[3] Dr.S.S Mantha,"Maximal Frequent Item set", International Journal of Computer Applications (0975 – 8887) Volume 10– No.3, November 2010.

[4] Sumathi k"an array based approach for mining maximal frequent itemsets "computational intelligence and computing research (iccic), 2010 ieee international conference on ICCIC.

[5] R.Divya Survey on AIS, Apriori and FP-Tree algorithms International Journal of Computer Science and Management Research Vol 1 Issue 2 September 2012

[6] Huanglin Zeng An Improved Algorithm of FP - tree Growth Based on Mapping Modeling (ICCASM 2010) V4-463

[7] Vaibhav Kant Singh and Vinay Kumar Singh "Minimizing Space Time Complexity by RSTDB a new method for Frequent Pattern Mining" To be appeared in Proceeding of the First International Conference on Intelligent Human Computer Interaction ,Allahabad,2009.

[8] Christie I. Ezeife and Min Chen Lecture Notes in Computer Science, , Volume 3129, Advances in Web-Age Information Management. 2004

[9] Han, J., J, Pei, Y, Yin and R, Mao, *2004*, *Mining* frequent patterns without candidate generations.

[10] *Han, J., J, Pei, Y, Yin and R, Mao*, 2004, Mining *frequent patterns without n improved* frequent pattern growth *method for* mining associatio*n rule.*

[11] Burdick Doug, Calimlim Manuel, and Gehrke Johannes, "A Maximal Frequent Item set Algorithm for Transactional Database", Proceedings of the 17th International Conference on Data Engineering, Heidelberg,Germany, pp . 443-452, April 2001.

[12] J Han, J Pei and Y Yin, "Mining frequent patterns without candidate generation", Proceedings of Special Interest Group on Management of Data, Dallas, pp. 1-12, May 2000.

[13] Agrawal R, Srikant S, Fast algorithms for mining association rules. In VLDB', 487-499, 1994 .